Parallel Voronoi Computation for Physics-Based Simulations

Julio Toss, João Comba, Bruno Raffin

Parallel Voronoi Computation for Physics-based Simulations

Julio Toss, João Comba, Bruno Raffin

1 Motivation

Voronoi diagrams are fundamental data structures in computational geometry with applications on different areas like on physics-based simulations. For non-Euclidean distances the Voronoi diagram must be performed over a grid-graph, where the edges encode the required distance information. The major bottleneck in this case is a shortest-path algorithm that must be computed multiple times during the simulation. We present a GPU algorithm for solving the shortest-path problem from multiple sources using a generalized distance function. Our algorithm was designed to leverage the grid-based nature of the underlying graph that represents the deformable objects. Experimental results report speed-ups up to 65x over a current reference sequential method. We discuss the performance benefits of this GPU algorithm for two Voronoi-based interpolation techniques: Sibson and Distance Ratio. Such approaches can be used for soft object simulation algorithms for real time physics engines.

2 Voronoi Diagrams

The Voronoi diagram is a classical partitioning of the space into closest-point regions. It has a vast application domain, being usually employed for answering proximity queries, for example in classical problems like *Finding Nearest Site*, *Facility Location*, *Motion Planning* and *Coverage in sensor networks*.

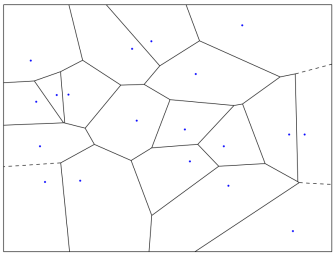


Figure 1. Voronoi diagram of a set of data points (in blue) in 2D *Euclidean* space. The space is partitioned into closest-point regions called *Voronoi cells*.

The Voronoi diagram is also a key for the "Natural Neighbor Interpolation"(NNI) method proposed by Sibson [15]. The Sibson's interpolation is a well-known method used for interpolating irregular spaced data with applications in several different fields such as medical imaging, meteorological or geological modeling [3], flow map reconstruction [1] and scattered data visualization [13]. Natural Neighbor based interpolations have also been applied to the field of solid mechanics by Sukumar et Al. [17] in the *Natural Element Method* (NEM) which uses Sibson and non-sibsonian (Laplace) interpolators to perform crack simulations.

Still in the field of physics simulation, Voronoi-based interpolations have also been applied to meshless simulation of complex deformable bodies [6]. In that work, the authors compute a Voronoi tessellation on a *non-Euclidean* space by using a discrete distance map which encodes "material-aware" distances that are biased according to the local rigidity inside the simulated body (see use case on Figure 2). Although the idea of a Voronoi partitioning of the space remains the same as in other classical applications, its computation here is fundamentally different. The distances are not defined on the Euclidean space, instead they rely on shortest paths computation over an implicit grid-graph. Computing this variant of Voronoi diagrams is therefore significantly more costly than the classical discrete Voronoi case. Using the Sibson's Interpolation method on a graph space actually requires to compute a shortest-path tree for each interpolated value queried.

A lot of effort has already been dedicated for improving the performance of Sibson's interpolation, particularly when interpolating on a discrete grid in the Euclidean space (*Discrete Sibson Interpolation*). A popular approach relies on the parallelization with Graphics Processing Units (GPUs) of the Discrete Voronoi diagram like on [3] for Digital Elevation Model (DEM) construction.

Early studies on the parallelization of the discrete Voronoi diagram already exploited the GPU parallel processing capabilities [8]. With the popularization of these architectures and evolution of the programming tools, like CUDA and OpenCL, other algorithms for Voronoi computation were proposed allowing a better utilization of their computing power [19, 14].

The graph variant of the Voronoi diagram, referred as *Graph Voronoi Diagram* [5], defines a vertex partitioning in a connected graph $G(V,E)$. Given a subset $S \subset V$ of *source* vertices, each vertex $v \in V$ is assigned to the partition P_i of the source vertex $s_i \in S$ that has the shortest path distance. Applications of this kind of Voronoi diagram arises in several network problems and social data analysis like, for instance, in community detection algorithms [4]. Sequential algorithms and complexity analysis for building Voronoi diagrams on graphs are discussed in [5].

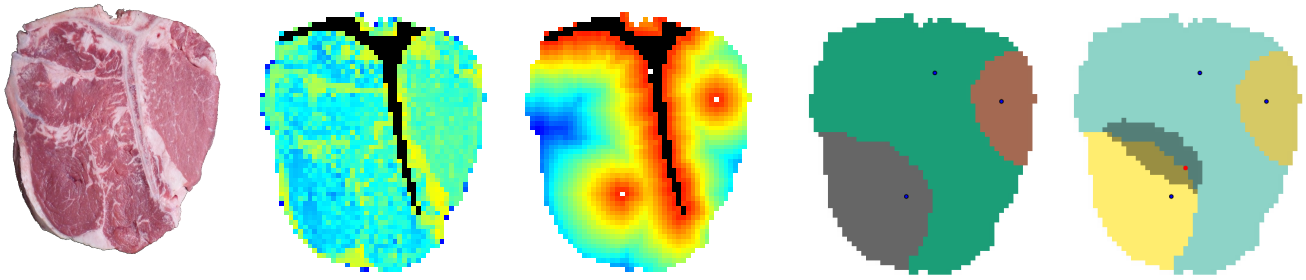
Parallel solutions for computing the Graph Voronoi have to deal with the shortest path problem. Dijkstra's algorithm implemented with a priority queue provides an efficient solution to this problem but is inherently sequential with lots o synchronizations. For graphs with grid topology, a first parallel algorithm to compute the Voronoi diagram was presented on [18]. The algorithm creates a distance map over a discrete image by computing the shortest paths from each pixel to the closest Voronoi seed (Figure 2(d)). The implementation is done on GPUs and takes advantage of the grid connectivity to leverage parallelism. This implementation has a straightforward application in the Voronoi-based interpolation methods used on [6] and other Natural Neighbor interpolations used in the SOFA real-time physics-based simulation framework [16].

In the following sections we will focus on the Discrete Voronoi diagram using graph distances (*geodesic*). We will show how this structure is used in the context of shape function computation for real-time physics-based simulations. We then present our parallel Voronoi implementation [18] applied to the Natural Neighbor Interpolation method.

3 Discrete Voronoi diagrams in SOFA

The Discrete Voronoi diagram is used in a real-time simulation framework called *SOFA* [16]. *SOFA* is a modular and extendable architecture that allows researchers of different fields related to physics-based simulation to implement and compare their own algorithms.

- *Julio Toss is a joint PhD student between Universidade Federal do Rio Grande do Sul (UFRGS / Brazil) and Univéristé Grenoble Alpes (UGA / France), e-mail:jtoss@inf.ufrgs.br*
- *João Comba is with UFRGS e-mail: comba@inf.ufrgs.br*
- *Bruno Raffin is with UGA/INRIA - France, e-mail: bruno.raffin@inria.fr*



(a) T-bone Steak (b) Stiffness (c) Material-aware distances (d) Voronoi Partitions (e) Natural Neighbors Interpolation

Figure 2. **Use case example:** (a) The T-bone steak from the *SOFA* dataset [6, 16] contains a mixture of flexible meat, softer grease and a rigid bone. As input we take the voxelized material map of stiffness values (b) and the coordinates of the simulation nodes. Distances to the nodes inside the object (c) are biased according to the stiffness values and used to compute the Voronoi diagram (d) of the nodes. This diagram will be used to compute the natural neighbors interpolation (e) on the other voxels of the domain.

3.1 Deformation using Shape Functions

In [6], Faure et Al. proposed a method for simulation of complex objects which are composed of mixed types of materials with different stiffness. This method relies on meshless models using sparse samples to capture the displacements at the simulation nodes which are then interpolated within the object using a novel *material-aware shape function*. This shape function uses a special distance metric that is scaled according to the local material rigidity of the simulated object (Figures 2(b) 2(c)). This technique allows to easily take into account the material heterogeneity during the simulation. The material properties of the object, like stiffness, is usually represented as a volumetric image of voxels containing the property values. To determine the material-aware distance, the shortest-path is computed over this 3D grid of voxels. Each voxel represents a vertex of a grid-graph with 26 neighbors, and the edges' weights are defined as a function of the stiffness of the adjacent voxels.

To understand the role of shape functions in numerical simulations consider the following steps:

1. The displacement of a deformable object is sampled at discrete locations called degrees of freedom (DoFs). Each DoF has a shape function associated that defines where and how it will influence other points in the object. The area of influence is often referred as the *support* of the shape function (Figure 3).

2. The goal is then to interpolate these sampled displacements within the rest of the object.

3. The displacement at a given point is interpolated as a weighted sum of the nodes displacements, where the weights are the values of the shape function for each DoFs influencing this point.

Finally, it remains the problem of defining how weights should be computed. This will tell how shape functions from different DoFs would be blended on the rest of the domain. Voronoi diagrams have been used in the so called *Voronoi Shape Functions* to compute these weights. In the following section we describe different schemes of interpolating nodal values using the Voronoi diagram generated from the simulation nodes.

3.2 Voronoi-based Interpolations

One of the most well-known interpolation methods based on the Voronoi diagram is the *Natural Neighbor Interpolation* (NNI) proposed by Sibson [15]. Consider the problem of finding neighbors in a set of non uniform distributed data points. By taking the Voronoi tessellation induced by these points, the *natural neighbors* are the data points whose Voronoi cells share a common frontier.

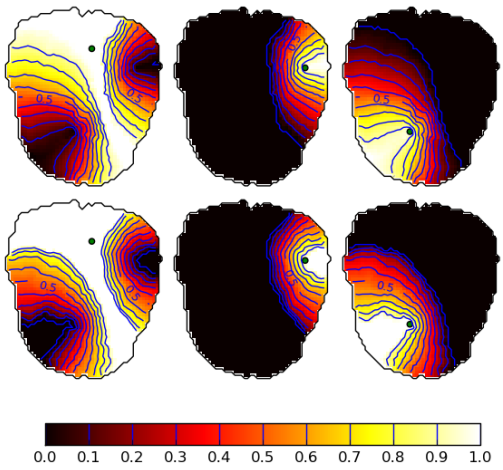


Figure 3. Shape-functions' weights for the 3 simulation nodes (from Figure 2) computed with two different interpolation methods: Sibson (top) and Distance Ratio (bottom). Weights are normalized starting at 1 at the node locations (Voronoi cell center) and decreases until it vanishes outside of the support.

The Sibson interpolation is defined as a ratio of areas in 2D (volumes in 3D). We insert the query point q in the initial Voronoi Tessellation of sample points. The interpolating weight of each data point is given by the ratio between the area stolen from the neighbor Voronoi cell and the area of the newly inserted Voronoi cell (Figure 2(e)).

The Laplace (non-Sibsonian) interpolation ([2] as cited in [17]) uses the same notion of natural neighbor but it computes the ratio between segments in 2D (areas in 3D). Instead of taking the area of the neighbor cells it uses the ratio between the length of the Voronoi frontier (line in 2D / facet in 3D) and the distance from q its natural neighbor nodes.

Both natural neighbors interpolation methods described above require the computation of a new Voronoi diagram for each query point q added to the input diagram of the data samples. This results in Q executions of the Voronoi diagram where Q is the interpolation resolution desired.

To reduce the complexity in terms of number of Voronoi diagrams computed per query point [6] propose a alternative interpolation method where the number of Voronoi diagrams computed is a constant

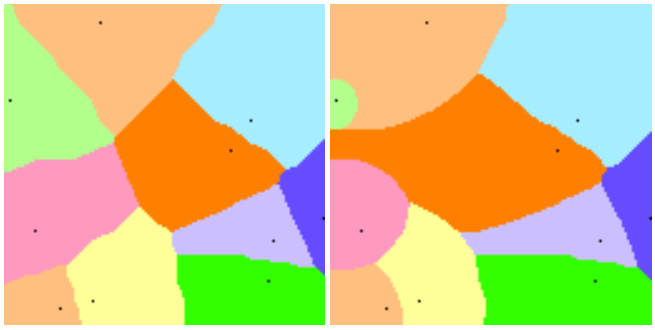


Figure 4. Comparison of Voronoi diagrams generated with the same set of seeds on two different material maps. Left: with an uniform stiffness. Right: with a stiffness gradient

factor of the amount of data samples S.

The Distance Ratio interpolation introduced by [6], applies a particular scheme that computes the ratio between the distance from the point to the Voronoi border and the distance to the node (center of the Voronoi cell). Although less formal guaranties on the properties were presented for this interpolant, this algorithm is implemented on SOFA [16] and shows good practical results, with the advantage of being more computationally efficient.

4 Parallel Voronoi Diagram Computation

As shown in the previous sections, the Voronoi diagram is an important building block for several other algorithms, in particular on the physics-based simulation domain. In real-time simulations, changes on the topology may modify the computed distances, for examples due to cut on the object. It is then necessary to be able to efficiently recompute the shape functions.

Parallel Voronoi computation on *euclidean distance* have been extensively studied on previous work [8, 19, 14]. However, such approaches cannot be directly applied on the physics simulation use cases described in this paper, where distance measures are actually shortest path computed on a graph. We show in this section that parallel Voronoi Diagrams on *geodesic distances* can be computed in parallel using GPUs.

Geodesic distance is closely related to the well studied single-source shortest-path (SSSP) problem from the graph theory domain. Parallel algorithms for solving the SSSP problem on general graphs have been proposed by [7, 9, 10, 12] . This solutions are usually based either on Dijkstra's or Bellman-Ford algorithm. Using these general graph algorithms for resolving the Voronoi Shape Functions in physics based simulation would imply an over-killing management of a full graph data-structure (like an adjacency matrix or list). Targeting efficiency, we presented in [18] a parallel algorithm using GPUs for computing the *graph-Voronoi* diagram on voxelized 3D grid. This solution is well suited for the physics simulations used in SOFA as it exploits the grid topology that implicitly represent edges. The solution is based on parallel wavefront expansions starting at each Voronoi seed. By using the massive amount of parallel threads of the GPU we are able to compute each Voronoi cell concurrently.

4.1 Parallel Voronoi Performance

We present here some experimental results of speed-up of the parallel Voronoi Computation. The experiments where conducted on a NVIDIA GPU GTX480 with 1.5 GBytes of global memory and 15 Multiprocessors with 32 cores each, totaling 480 CUDA cores. The speed-up presented are related to the base sequential version from *SOFA* [16] and executed on an Intel CoreTM i7 CPU model 930 with 4 cores running at 2.89Ghz and 12 GB memory .

The benchmark consists in computing the Voronoi diagram on a 3D volume for a given set of randomly distributed data points. The dataset used varies in *volume dimensions*, *distance map distribution* and *number of seeds*. Figure 4 shows an example of two Voronoi diagrams computed with same dimensions and seeds but using different distance distribution. The right diagram was computed considering an image with a gradient of stiffness increasing from right to left whereas the left one has constant stiffness. These two distances distributions are referred as *Gradient* and *Constant* on the bar plots of Figure 5. Note that the distance between two neighbor voxels is given by a function of the stiffness between them. As seen in Figure 4 the shape of the Voronoi diagram greatly depends on the stiffness properties of material map of the object simulated .

In the parallel algorithm implemented each CUDA thread processes a single voxel. With larger input volumes more parallelism is exposed, therefore we note a increase of speed-up (Figure 5). At 256^3 the speed-up for the constant topology slightly decreases. We attribute this to an overhead of scheduling an excessive number of idle threads. This happens because with a volume of 256 and only 10 seeds the Voronoi diagram becomes overly sparse.

The amount of parallel work available also increases with the number of seed in the Voronoi diagram (Figure 6). When many Voronoi cells are being computed concurrently more threads are active at the same time.

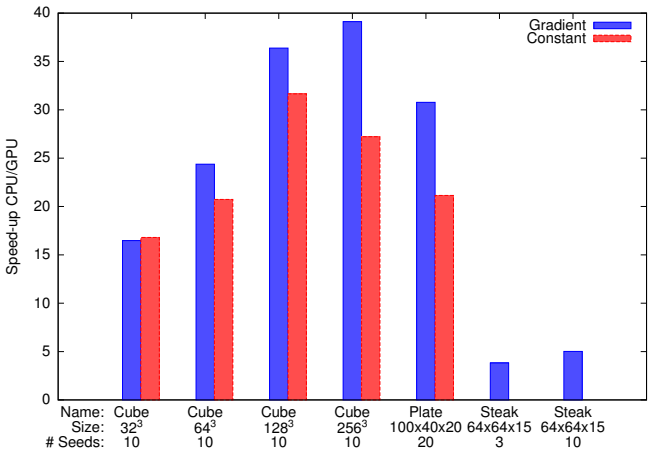


Figure 5. Speed-up for different input sizes. Gradient and constant topologies are presented for synthetic benchmarks only. Steak's topology corresponds to the dataset shown in the use case of Figure 2.

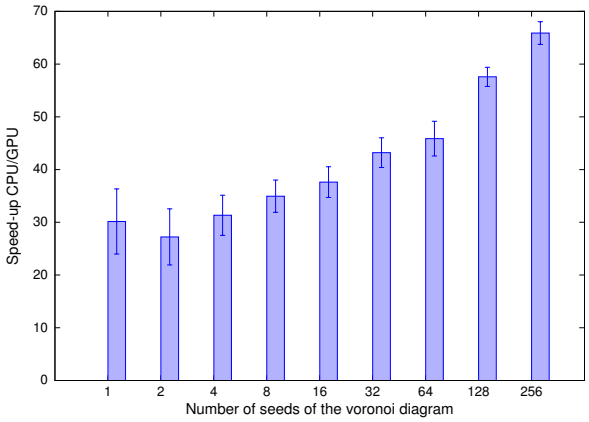


Figure 6. Average speed-up when increasing the number of seeds of the Voronoi diagram. The instance used has a volume size of 128^3 and a gradient material map.

5 Parallel Natural Neighbor Interpolations

In the classical Natural Neighbor Interpolation, each queried point is inserted, one at a time, to the Voronoi diagram of initial data samples. For each seed added, the initial diagram is updated to generate the new Voronoi cell which will then be used to compute the interpolation.

As seen in the experimental results, the number of seeds computed in the Voronoi diagram has a big impact on the amount parallelism that will be exposed. Computing a single Voronoi cell in parallel reduces the possibilities of parallelization. This situation is even worse when we consider to update an existing Voronoi diagram with the addition of a new seed (we refer to this as the *query seed*). In this case, only a limited region (around the *query seed*) would be recomputed (e.g. Figure 2(e)). Performing NNI interpolation simply as sequence of (parallel) Voronoi computations on the GPU doesn't pay-off the overheads of memory transfers and thread scheduling inherent to this architecture.

The evident strategy to generate interpolated values over a grid would be to perform multiple queries concurrently in parallel. This approach was used in [3] to generate a interpolation of regular grid using the assumption that every sample has limited radius of influence. This allowed to decompose the domain in independent blocks where queries could be answered in parallel batches. In [13], Park et Al. propose a more efficient implementation of Sibson's interpolation on raster images. Their method avoid the explicit construction of a new Voronoi diagram for each query point. Instead they use a Kd-tree structure to find the closest seed to the current query point and use this distance as a radius of influence to increment the interpolation weight. Again, these techniques make assumptions that are valid for euclidean space but not trivially generalized for geodesic (graph) distances.

Parallelization can still be implemented for Natural Neighbor Interpolation over graph spaces if we duplicate some data structures. More precisely, for each NNI query, we copy the input Voronoi diagram of data sample and update it with the addition of a new seed at the coordinates of this query.

Table 1 shows the computation time spent for the parallel Sibson algorithm on a NVidia Tesla K40 GPU. The interpolation is performed over a uniform grid of dimensions $100 \times 40 \times 20$ (80000 voxels) and 20 data points. The GPU algorithm performs batches of parallel NNI queries in sequence iteratively until all the grid is computed. We show the average amount of time spend by each batch of parallel queries and the total time for interpolating the whole grid. Note that the parallel version manages to amortize the overhead when more than 10 NNI queries are done in parallel.

Table 1. Computation time for parallel NNI queries. Avg. Sibson Query corresponds to time spent for querying a whole batch.

Batch Size	Avg. Sibson Query (ms)	Total time (ms)
CPU 1 (seq)	1.003	81057.226
GPU 1	9.745	779657.756
GPU 10	12.413	99309.125
GPU 100	29.537	23629.870
GPU 150	34.807	18587.130

6 Conclusion and Future Work

In this work we review a non-usual application of a classical geometrical structure: the Voronoi diagram applied to meshless simulations. We show that the parallelization with GPUs can be used to speed-up the computation of the Voronoi diagram variant required by this domain.

Considering the characteristics of this application - interpolation on a discrete grid and non-euclidean distances - we build upon our parallel Voronoi algorithm to show how it can be used to parallelize the well-know Natural Neighbor Interpolation method.

The parallel Voronoi implementation presented has a valuable application in soft object simulation methods. It provides a performance solution to the physics-based simulation community that wants to employ Voronoi shape-functions in their meshless simulations.

Finally, recent work have proposed to also use Voronoi Shape-functions on grids with "extended" connectivity (*non-manifold grids* [11]) which would allow to represent objects with more complex topologies in meshless frameworks. Possible extensions of this work will consider the application of our parallel algorithm in these new domains.

7 Acknowledgment

We would like to thank project *Capes/Cofecub 764/13* for the financial support.

References

[1] S. S. Barakat and X. Tricoche. Adaptive refinement of the flow map using sparse samples. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2753–2762, 2013.

[2] V. V. Belikov, V. D. Ivanov, V. K. Kontorovich, S. A. Korytnik, and A. Y. Semenov. The non-Sibsonian interpolation: A new method of interpolation of the values of a function on an arbitrary set of points. *Computational mathematics and mathematical physics*, 37(1):9–15, 1997.

[3] A. Beutel, T. Mølhave, and P. K. Agarwal. Natural neighbor interpolation based grid DEM construction using a GPU. *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '10*, page 172, 2010.

[4] D. Deritei, Z. I. Lázár, I. Papp, F. Járai-Szabó, R. Sumi, L. Varga, E. R. Regan, and M. Ercsey-Ravasz. Community detection by graph Voronoi diagrams. *New Journal of Physics*, 16(6):063007, jun 2014.

[5] M. Erwig. The graph Voronoi diagram with applications. *Networks*, 36(3):156–163, 2000.

[6] F. Faure, B. Gilles, G. Bousquet, and D. K. Pai. Sparse meshless models of complex deformable solids. *ACM SIGGRAPH 2011 papers on - SIGGRAPH '11*, page 1, 2011.

[7] P. Harish, V. Vineet, and P. J. Narayanan. Large Graph Algorithms for Massively Multithreaded Architectures. Technical Report IIIT/TR/2009/74, International Institute of Information Technology Hyderabad, 2009.

[8] K. E. Hoff III, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized Voronoi diagrams using graphics hardware. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 277–286, 1999.

[9] S. Kumar, A. Misra, and R. S. R. S. Tomar. A modified parallel approach to Single Source Shortest Path Problem for massively dense graphs using CUDA. *2011 2nd International Conference on Computer and Communication Technology (ICCCT-2011)*, pages 635–639, sep 2011.

[10] K. Madduri, D. Bader, J. Berry, and J. Crobak. Parallel shortest path algorithms for solving large-scale instances. In *9th DIMACS Implementation Challenge – The Shortest Path Problem*, pages 1–39, DIMACS Center, Rutgers University, Piscataway, NJ, 2006.

[11] P.-L. Manteaux, W.-L. Sun, F. Faure, M.-P. Cani, and J. F. O'Brien. Interactive Detailed Cutting of Thin Sheets. In *ACM SIGGRAPH Motion in Games (MIG)*, pages 1–8, 2015.

[12] H. Ortega-Arranz, Y. Torres, D. R. Llanos, and A. Gonzalez-Escribano. A new GPU-based approach to the Shortest Path problem. In *2013 International Conference on High Performance Computing & Simulation (HPCS)*, pages 505–511. IEEE, jul 2013.

[13] S. W. Park, L. Linsen, O. Kreylos, J. D. Owens, and B. Hamann. Discrete sibson interpolation. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):243–252, 2006.

[14] G. Rong and T. S. Tan. Jump flooding in GPU with applications to Voronoi diagram and distance transform. *Proceedings of the 2006 symposium on Interactive 3D . . .* , page 109, 2006.

[15] R. Sibson. A Brief Description of Natural Neighbor Interpolation. *Interpreting Multivariate Data*, 21:21–36, 1981.

[16] P. SOFA. Simulaton Open Framework Architecture. `http://www.sofa-framework.org/`, jan 2016.

[17] N. Sukumar. Voronoi cell finite difference method for the diffusion operator on arbitrary unstructured grids. *International Journal for Numerical Methods in Engineering*, 57(1):1–34, 2003.

[18] J. Toss, J. L. D. Comba, and B. Raffin. Parallel Shortest Path Algorithm for Voronoi Diagrams with Generalized Distance Functions. In *27th SIBGRAPI - Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 212–219, 2014.

[19] O. Weber, Y. S. Devir, A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Transactions on Graphics*, 27(4):1–16, oct 2008.